Suunnitteludokumentti

Kohahdus

Helsinki 11.10.2006

Ohjelmistotuotantoprojekti

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Kurssi

581260 Ohjelmistotuotantoprojekti (6 ov)

Projektiryhmä

Taro Morimoto, Projektipäällikkö
Tuomas Palmanto, Vaatimusmäärittelyvastaava
Mikko Kinnunen, Suunnitteluvastaava
Markus Kivilä, Koodivastaava
Jari Inkinen, Testausvastaava
Paula Kuosmanen, Dokumenttivastaava

Asiakas

Teemu Kerola

Johtoryhmä

Sanna Keskioja

Kotisivu

`http://www.cs.helsinki.fi/group/kohahdus`

Versiohistoria

Versio	Päiväys	Tehdyt muutokset
1.0	21.1.2004	Ensimmäinen versio

Sisältö

1 Johdanto **1**

2 Sanasto **1**

3 Toiminnot **2**

3.1 Rekisteröityminen 2
3.2 Käyttäjätietojen muokkaaminen 2
3.3 Kirjautuminen 2
3.4 Kurssien ja tehtävien valinta (opettaja) 2
3.5 Tehtävän määrittely (opettaja) 3
3.6 Kurssin statistiikka (opettaja) 3
3.7 Käyttäjän tiedot (opettaja) 3
3.8 Tehtävien valinta (opiskelija) 3
3.9 Tehtävään vastaaminen (opiskelija) 3

4 Arkkitehtuurisuunnitelma **4**

5 Tietokanta **4**

5.1 Tasktype 5
5.2 Task 5
5.3 Course 5
5.4 Module 5
5.5 Taskinmodule 5
5.6 Attributevalues 6
5.7 EAuser 6
5.8 Storedanswer 6
5.9 Studentmodel 6
5.10 Taskattributes 6
5.11 Pluginparameters 6
5.12 Pluginparamattributes 7

6 Luokat **7**

6.1 Criterion 7

6.2 DBHandler 8
6.3 Task 9
6.4 User 11
6.5 Course 13
6.6 TitoAnalyzer 13
6.7 TitoDisplayer 13
6.8 TitoState 14

7 JSP sivujen sekvenssikaaviot 14
7.1 Register.jsp 14
7.2 Login.jsp 14
7.3 TeacherTaskList.jsp 14
7.4 TaskList.jsp 14
7.5 Report.jsp 14
7.6 UserInfo.jsp 14
7.7 Composer.jsp 15
7.8 TeacherTaskList.jsp 15
7.9 Answer-servlet 15

1 Johdanto

TitoTrainer on järjestelmä automaattisesti tarkastettavien TTK-91-konekielen harjoitustehtävien luomiseen ja ratkaisemiseen. Järjestelmä on tarkoitettu käytettäväksi opetuksen tukena, opetettaessa Tietokoneen toiminta -kurssia. Tietojenkäsittelytieteen opettajat voivat tehdä järjestelmään uusia tehtäviä ja määritellä kuinka ne tarkastetaan automaattisesti. Tietokoneen toiminta -kurssin opiskelijat ja kurssin tehtävistä kiinnostuneet itseopiskelijat voivat ratkaista tehtäviä ja saada palautetta niiden onnistumisesta.

Painopiste projektissa on opettajan ja opiskelijan käyttöliittymillä. Käyttöliittymistä tehdään mahdollisimman selkeät ja helppokäyttöiset. Opettajan käyttöliittymän avulla määritellään tehtävät parametreineen ja kuinka opiskelijan ratkaisun oikeellisuus tarkistetaan, sekä nähdään statistiikkatietoja opiskelijoiden suorituksista. Opiskelijan käyttöliittymän avulla opiskelija valitsee tehtäviä, syöttää niiden ratkaisut ja saa palautetta vastauksensa oikeellisuudesta.

Projekti käyttää valmiina olevaa Titokone-simulaattoria harjoitustehtävien ratkaisemiseen. Projektin rakennetaan olemassa olevaan eAssari-kehykseen, joka sopii geneeriseen automaattisesti tarkistettavien tehtävien määrittelyyn ja toteutukseen. eAssarin keskeneräisyydestä johtuen kehystä tullaan muokkaamaan tuotteella asetettujen vaatimusten täyttämiseksi.

Järjestelmä toteutetaan Java-kielellä ja järjestelmän käyttö vaatii, että selain tukee JavaScript-kieltä ja CSS-tyylitiedostoja. Järjestelmä toimii uusimmilla Firefox ja Internet Explorer-selaimilla.

2 Sanasto

TTK91=Auvo Häkkisen kehittämä ohjelmointikieli, joka läheisesti muistuttaa symbolista konekieltä.

KOKSI=Auvo Häkkisen kirjoittama konekielisimulaattori, joka toteuttaa TTK-91-kielen.

Kohahdus=Syksyn 2006 ohjelmistotuotantoprojekti (tämä projekti)

TitoTrainer=Projektimme tuotos

Järjestelmä=TitoTrainer

Ohjelma=Opiskelijan kirjoittama TTK91-ohjelma, eli vastaus johonkin tehtävään

eAssari=Tietokantapohjainen ympäristö ohjelmallisesti tarkastettavien harjoitus- ja koetehtävien suorittamiseen

Titokone=Koski-nimisen Ohjelmistotuotantoprojektiryhmän vuonna 2004 rakentama järjestelmä konekielisten ohjelmien kääntämiseen ja suorittamiseen.

Koski=Vuoden 2004 Ohjelmistotuotantoprojekti joka rakensi konekielen simulaattorin ja debug-ympäristön, eli Titokoneen

Koskelo=Vuoden 2004 Ohjelmistotuotantoprojekti, joka integroi Titokoneen ja eAssari-

kehyksen yhteen. Ratkaisusta ei tullut kuitenkaan käyttökelpoista, eikä sitä ole otettu käyttöön.

Kriteeri=Sääntö jonka mukaan tehtävän oikeellisuus tarkistetaan. Kriteereitä voi olla monta yhdelle tehtävälle.

Aihepiiri=Tehtävälle täytyy määritellä aihepiiri, johon tehtävä kuuluu.

3 Toiminnot

Tämä luku kuvaa järjestelmän HTML-sivut ja sen, mitä toimintoja ne tarjoavät käyttäjälle.

3.1 Rekisteröityminen

Rekisteröintisivulla opiskelija rekisteröityy järjestelmän käyttäjäksi. Rekisteröinti vaatii seuraavat tiedot: etunimi, sukunimi, käyttökieli, sähköposti, opiskelijanumero tai henkilötunnus, sekä haluttu käyttäjätunnus ja salasana

3.2 Käyttäjätietojen muokkaaminen

Käyttäjätietosivulla sisäänkirjautunut käyttäjä voi muokata rekisteröintitietojaan, poislukien tunnus joka on valittu rekisteröidyttäessä. Käyttäjätietojen muokkaamisen ei tarvitse toimia opettajien tunnusten kanssa, mutta odotettavissa on, että opettajatunnusten tukeminen ei aiheuta minkäänlaista lisätyötä.

3.3 Kirjautuminen

Kirjautumissivulla käyttäjä syöttää tunnuksensa, salasanansa ja pudotusvalikosta kurssin nimi. Kurssivalinta määrittää, että minkä kurssin suorituksiin opiskelijan tekemät vastaukset tallennetaan. Opettajan ollessa kyseessä kurssivalinnalla ei ole merkitystä. Jos kirjautuminen onnistuu, näytetään tehtävälistaus. Jos kirjautuminen epäonnistuu, käyttäjä palautetaan kirjautumissivulle.

3.4 Kurssien ja tehtävien valinta (opettaja)

Kirjautumisen jälkeen opettajalle näytetään kurssilistaus josta opettaja voi siirtyä kurssin statistiikka-sivulle. Kurssilistauksesen alla on myös lomake uuden kurssin luomista varten.

Samalla sivulla näytetään myös tehtävälista. Listassa näytetään tehtävän kieli, aihepiiri ja nimi, sekä viimeinen muokkauspäivä ja muokkaaja. Listattuja tehtäviä voi muokata ja poistaa, sekä käynnistää uuden tehtävän luomisen.

3.5 Tehtävän määrittely (opettaja)

Tehtävänmäärittelysivulla opettaja voi luoda uuden tehtävän tai muokata olemassa olevaa tehtävää. Tehtävälle määritellään nimi, tehtävänanto, tarkastustyyppi (malliratkaisu/kiinteät arvot), tehtävätyyppi (täyttötehtävä/ohjelmointitehtävä), sekä tarkastuskriteerit ja kriteerien palautteet.

3.6 Kurssin statistiikka (opettaja)

Kurssin statistiikkasivulla opettajalle näytetään lista kaikkien kurssin tehtäviä suorittaneiden opiskelijoiden suorituksista. Listassa näytetään opiskelijan nimi, opiskelijanumero tai henkilötunnus, sekä onnistuneesti ratkaistujen tehtävien määrä. Lista on helposti tulostettavassa muodossa.

Listassa opiskelijan nimi toimii linkkinä ko. opiskelijan käyttäjätietosivulle.

Kurssin statistiikka toteutetaan toisessa kehitysiteraatiossa.

3.7 Käyttäjän tiedot (opettaja)

Käyttäjän statistiikkasivulla opettajalle näytetään opiskelijan tarkemmat tiedot. Listassa näkyy salasanaa lukuunottamatta kaikki rekisteröintitiedot.

Käyttäjän tiedot toteutetaan toisessa kehitysiteraatiossa.

3.8 Tehtävien valinta (opiskelija)

Kirjautumisen jälkeen opiskelijalle näytetään tehtävälista, josta opiskelija voi valita tehtävän ratkaistavaksi. Listassa näytetään tehtävän kieli, aihepiiri ja nimi, sekä tieto onko opiskelija yrittänyt jo tehtävän suoritusta ja onko yritys onnistunut.

3.9 Tehtävään vastaaminen (opiskelija)

Opiskelijan ratkaistavaksi valitsemasta tehtävästä näytetään nimi ja tehtävänanto, mahdollinen valmis koodi mikäli kyseessä on täyttötehtävä, sekä tekstialue johon opiskelija kirjoittaa vastauksensa. Lisäksi on linkki jolla pääsee takaisin tehtävälistaukseen.

Kun opiskelija on kirjoittanut vastauksensa, hän painaa "Suorita-nappia joka vie takaisin tehtävävastaussivulle. Sivulla näytetään samat tiedot kuin alussa, mutta lisäksi näytetään palautetta tehtävän ratkaisun onnistumisesta/epäonnistumisesta.

Tehtävään vastaaminen toteutetaan toisessa kehitysiteraatiossa.

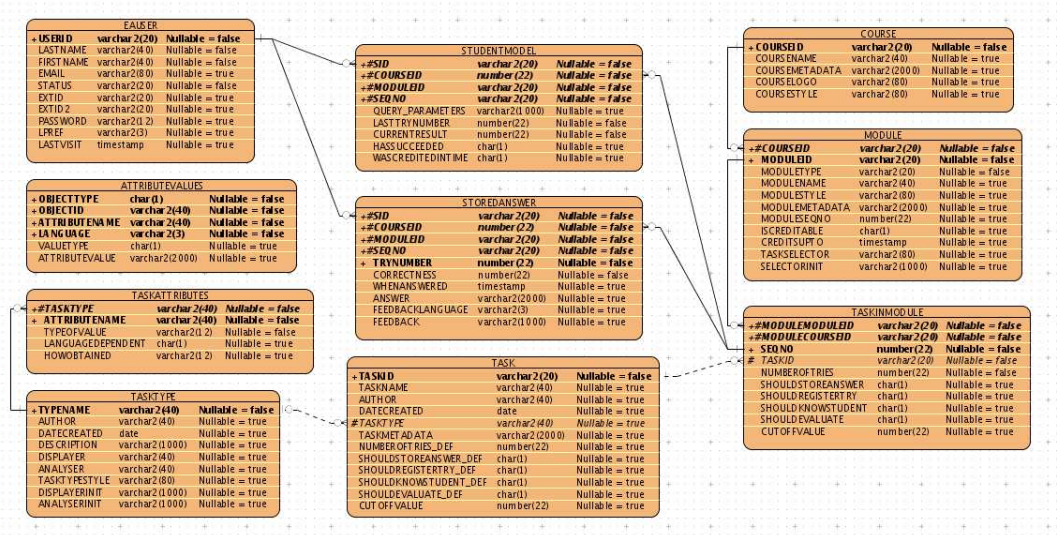
4 Arkkitehtuurisuunnitelma

Järjestelmä on jaettu kahteen osajärjestelmään, Titokone ja eAssari/TitoTrainer. Ideaalitapaus olisi ollut kolme osajärjestelmää, jossa eAssari on TitoTraineria kutsuva sovelluskehys. Tämä malli ei kuitenkaan ole mahdollinen, johtuen eAssarin keskeneräisyydestä. Lopputuotteelle asettujen vaatimusten täyttäminen vaatii huomattavia lisäyksiä eAssariin, mm. puuttuvat kurssien ja käyttäjien hallinta täytyy tehdä. Kireän aikataulun vuoksi eAssaria ei yritetä säilyttää geneerisenä kehyksenä, vaan muokkauset tehdään vain Kohahdus projektin tarpeita ajatellen. Käytännössä eAssarista tehdään siis ns. code fork, eli synnytetään täysin TitoTrainer-spesifinen eAssari-versio, jonka ei taata olevan yhteensopiva geneerisen eAssarin kanssa.

Ainoa osuus eAssarista jota pyritään käyttämään sen alkuperäisen arkkitehtuurin mukaan, on tehtävien tarkistus eli Answer-servlet ja Analyzer sekä Feedback rajapinnat. Tietokannan rakennetta ei myöskään muuteta, mutta kaikkia tauluja ei käytetä (kts. tämän dokumentin luku Tietokanta).

Titokonetta käytetään sen TTK91-rajapintojen kautta, joita muokataan tarpeen mukaan jotta vaatimuksissa esitetyt tehtävien tarkistukset ja suorituksenaikaiset tilastot saadaan toteutettua. Titokoneen käytön ja muokkauksen yksityiskohdat jätetään toisen iteraation asioiksi.

5 Tietokanta



Kuva 1: Tietokantakaavio

Tietotokannan rakenteen määrää eAssari kehys. Rakenne on kuvattu liitteessä 1. Ajanpuutteesta johtuen module-tietokantataulua ei tulla käyttämään sen mahdollistamassa laajuudessa. Kyseisestä taulusta luodaan vain yksi rivi kantaan, johon sitten viitataan muista tauluista. Lisäksi jokainen tehtävä sisältyy jokaiseen kurssiin. Eli kun tehtävä luodaan,

niin se liitetään jokaiseen kurssiin. Lisäksi kun kurssi luodaan, niin jokainen tehtävä lisätään kyseiseen uuteen kurssiin. Tämä linkitys tapahtuu "taskinmodule-tietokantataulussa.

Tietokantaa käytetään aina DBHandler-luokan kautta, joka piilottaa tietokantarakenteen yksityiskohdat. Eli SQL-lauseita tulee vain ks. luokkaan eikä muualle koodiin. DBHandler-luokasta voidaan luoda instanssi staattisen DBHandler.getInstance() -metodin avulla. Tämä tarkoittaa sitä, että ko. luokkaa voidaan käyttää virtuaalikoneen sisällä mistä vaan.

5.1 Tasktype

Tasktype on eAssari-kehyksen tarvitsema taulu, joka määrittelee tehtävätyypin käyttämät Analyzer ja Displayer luokat. Tauluun olisi mahdollista tallentaa myös muuta tehtävätyyppikohtaista tietoa, mutta ainakaan ensimmäisen iteraation kohdalla tälle ei ole näkyvissä tarvetta.

5.2 Task

Task määrittelee tehtävän. Itse task-tauluun tallennetaan lähinnä tehtävän nimi ja luojan nimi. Tehtävän kriteerit, kuvaus, palautteet yms lisätään yleiskäyttöiseen attributevalues-tauluun.

5.3 Course

Määrittelee kurssin nimen ja tunnisteen. Tähän tauluun viitataan useista muista tauluista, jotta esimerkikisi opiskelijoiden suoritukset voidaan kirjata tietylle kurssille.

5.4 Module

Luotuja tehtäviä ei liitetä suoraan kursseihin, vaan moduuleihin ja moduulit liitetään kursseihin. Kurssiin voi liittää useita moduuleja, mutta yhtä moduulia ei voi lisätä useaan kurssiin. TitoTrainer ei käytä moduuleja siinä laajuudessa kuin tietokannan rakenne mahdollistaisi, vaan toteutuksen ja käyttöliittymien yksinkertaistamiseksi jokaiselle kurssille luodaan tarkalleen yksi moduuli johon lisätään kaikki tehtävät.

5.5 Taskinmodule

Yhdistää tehtävän kurssiin ja moduuliin. Normaalitapauksessa siis tehtävä lisättäisiin jonkin kurssin johonkin moduulin luomalla rivi taskinmodule-tauluun.

TitoTrainerin tapauksessa on määritelty, että kaikki tehtävät kuuluvat kaikkiin kursseihin, joten kun opettaja luo tehtävän, TitoTrainer luo automaattisesti yhtä monta taskinmodule riviä kuin on kursseja. Samoin kun luodaan uusi kurssi, sille luodaan automaattisesti

moduuli ja kaikki olemassa olevat tehtävät lisätään ko. moduuliin. Näin ollen aina pitää paikkansa, että Taskinmodule rivien määrä = Task rivien määrä * Course rivien määrä.

5.6 Attributevalues

Yleiskäyttöinen varasto jonne voi tallentaa mitä tahansa lisätietoa koskien mitä tahansa muuta taulua. Tänne tallennetaan tehtävien kriteerit, palautteet, aihepiiri, esimerkkiratkaisu, ja lähes kaikki muu tieto tehtävästä nimeä lukuunottamatta. Lisäksi voidaan tallentaa esimerkiksi lisätietoja kursseista jos tarvetta ilmenee.

5.7 EAuser

Taulu kaikkien käyttäjätietojen tallentamiseen. Sekä oppilaat ja opettajat tallennetaan tänne, erona näillä on vain status-sarakkeen arvo.

5.8 Storedanswer

Kun opiskelija vastaa tehtävään, vastaus sekä tiedot sen oikeellisuudesta tallennetaan tähän tauluun.

5.9 Studentmodel

Jokaista tehtävän vastausyritystä kohden luodaan storedanswer-tauluun yksi rivi. Studentmodel-taluun tallennetaan yksi rivi per tehtävä, vaikka vastausyrityksiä olisi useampia. Rivillle tallennetaan tieto siitä, onko tehtävä suoritettu onnistuneesti ja kuinka monta yritystä tehtävään on käytetty. Nämä tiedot olisi mahdollista päätellä koostefunktioden avulla storedanswer-taulusta, mutta ilmeisesti tehokkuuden parantamiksi on luotu erillinen koostetaulu.

5.10 Taskattributes

Task-attributes on eAssarissa tarkoitettu kuvaamaan tehtävien attribuuttien rakennetta. eAssari ei kuitenkaan missään vaiheessa itse hyödynnä tätä taulua, eikä TitoTrainerkään sitä tarvitse, joten taulu jää täysin käyttämättä.

5.11 Pluginparameters

Tämän taulun tarkoitus on jäänyt täysin hämärän peittoon, eAssari ei missään kohtaa koodissaan viittaa mihinkään plugineihin. TitoTrainerillä ei myöskään ole tälle taululle mitään käyttöä, joten se jää täysin käyttämättä.

5.12 Pluginparamattributes

Kuten pluginparameters, jää käyttämättä.

6 Luokat

6.1 Criterion

public abstract class Criterion

Base class for all criterion types. The many different types of criteria in TitoTrainer are all used via the interface defined here. The analyzer component does not know the details of different Criterion sub-classes. Only the composer used for creating and modiying Tasks is even aware that differnt types of criteria exist.

`public String getPositiveFeedback(TitoState studentAnswer)`
Return the positive feedback string of this Criterion. Criterion types that also evaluate quality of the answer should overide this method so they can return a different string depending on the quality of the student's answer.

`public String getNegativeFeedback()`
Return the negative feedback string of this Criterion

`public boolean isSecretInputCriterion()`
Return true if this criterion is to be used with secret input

`public String serializeToXML()`
Return a serialized copy of this Criterion in XML-format

`public abstract boolean meetsCriterion(TitoState studentAnswer, TitoState modelAnswer);`
Return true if student's answer meets the condition(s) of this criterion. Criterion types that also evaluate quality of the answer must return <code>true</code> if the answer fullfills the passing requirement, even if answer was deemed low quality.

`protected abstract String serializeSubClass();`
Serialize non-static data-members of Criterion sub-class to XML format. The subclass can freely decide the names of the XML tags. The abstract Criterion class will handle the serialization of its data-members. The serialized string is stored in the eAssari database in a 2000-char field so subclasses should try to keep the tags and data short (without being cryptic).

`protected abstract void initSubClass(String serializedXML);`
Initialize non-static data-members of this Criterion subclass instance using the serialized representation returned by <code>serializeToXML()</code>. The data-member of the abstract Criterion class will have already been deserialized when this method is called.

`public static Criterion deserializeFromXML(String xml)`
Instantiate new Criterion object using the serialized form Xml

`protected static String toXML(String tagname, boolean value)`
Serialize boolean value to XML string. Helper function for serializeSubClass()

`protected static String toXML(String tagname, String value)`
Serialize String value to XML string. Helper function for serializeSubClass()

`protected static String toXML(String tagname, int value)`
Serialize integer value to XML string. Helper function for serializeSubClass()

`protected static boolean parseXMLBoolean(String XML, String tagname)`
Deserialize boolean value from XML string. Helper function for initSubClass()

`protected static String parseXMLString(String XML, String tagname)`
Deserialize String value from XML string. Helper function for initSubClass()

`protected static int parseXMLInt(String XML, String tagname)`
Deserialize integer value from XML string. Helper function for initSubClass()

6.2 DBHandler

public class DBHandler

Singleton class used for database interactions. Each public method of DBHandler class encapsulates one database transaction, and thus may cause multiple inserts/updates/removes with one call. The atomicity of the operations is quaranteed by using the transaction model provided by the SQL standard.

`private DBHandler()`
Prevent youside instatiation by giving constructor private scope

`public static synchronized DBHandler getInstance()`
Return DBHandler instance

`public Task[] getTasks()`
Return all tasks

`public Task getTask(String taskID)`
Return task identified by taskID

`public Criterion[] getCriteria(Task task)`
Return the criteria of Task task

`public void createTask(Task task, Criterion[] criteria)`
Add new task to task database. The insert will affect all courses. This operation will also create the criteria for the task

`public void updateTask(Task task, Criterion[] criteria)`
Update existing task. The update will affect all courses. This operation will also update the criteria for the task.

`public void removeTask(Task task)`
Remove task from task database (and thus all courses). This will also remove all stored answers of the task.

`public User[] getUsers(Course c) throws SQLException`
Return all users who have attempted to solve at least one task of Course c

`public User getUser(String userID, String password) throws SQLExcepti`
Return user identified by userID

`public User getUser(String userID) throws SQLException`
Return user identified by userID

`public void createUser(User user)`
Add new user to user database

`public void updateUser(User user)`
Update existing user

6.3 Task

public class Task

Modify existing Task class by adding the following methods

`public String getName()`
Return the name of this task

`public void setName(String name)`
Set the name of this task

`public String getAuthor()`
Return name of the last person who has modified this task

`public void setAuthor(String name)`
Set "last task modification by"attribute to Name, set last-modification-timestamp to current data and time

`public Date getModificationDate()`
Return the date and time this task was last modified

`public String getModelAnswer()`
Return code of the model answer provided by teacher

`public void setModelAnswer(String code)`
Set model answer code

`public boolean isValidateByModel()`
Return true if this task is to be validated by comparing results of the student's answer to results of teacher's answer

`public void setValidateByModel(boolean useModel)`
Set the validation method of this task

`public String getCategory()`
Return the task category of this task

`public void setCategory(String category)`
Set task category of this task

`public boolean isFillInTask()`
Return true if this is a fill-in task

`public void setFillInTask(boolean fillIn)`
Set this task as fill-in or create-full-program

`public String getFillInPreCode()`
Return the code that is prepended before student's code in a fill-in task

`public void SetFillInPreCode(String code)`
Set the code that is prepended before student's code in a fill-in task

`public String getFillInPostCode()`
Return the code that is appended to student's code in a fill-in task

`public void SetFillInPostCode(String code)`
Set the code that is appended to student's code in a fill-in task

`public String getDescription()`
Return the description (tehtävänanto) of this task

`public void setDescription()`
Set description (tehtävänanto) of this task

6.4 User

public class User

Modify existing User class by adding the following methods

```
public static final String STATUS_STUDENT = "student";
public static final String STATUS_TEACHER = "teacher";
```

`public User()`
Construct unitialized User object

`public User(String userID)`
Construct unitialized User object with userid

`public void setLastName(String lastname)`
Set last name of this user

`public void setFirstName(String firstname)`
Set first name of this user

`public void setEmail(String email)`
Set email address of this user

`public void setUserID(String userID)`
Set userID of this user

`public String getStudentNumber()`
Return student number of this user. This identifier maps to <code>aeuser.extid</code> in the database

`public void setStudentNumber(String studentnum)`
Set student number of of this user

`public String getSocialSecurityNumber()`
Return social security number of this user. This identifier maps to <code>aeuser.extid2</code> in the database

`public void setSocialSecurityNumber(String ssn)`
Set social security number of this user

`public void setPassword(String pass)`
Set password of this user to Pass

`public void setLanguage(String lang)`
Set the preferred language of this user. The language is either "EN"or "FI

`public String getStatus()`
Return status of this user

`public boolean isTeacher()`
Return true of this user has the privelidges to add/remove/modify tasks and browse user statistics

`public void setStatus(String status)`
`throws IllegalArgumentException if Status is not a valid Status string`
Set user status (teacher / student)

`public boolean isValid()`
Test validity of this user object. The object is considered valid if all data members are set with non-empty values.

6.5 Course

public class Course

Simple class for holding basic course information

`public Course(String name, String id)`
Create new Course instance using the specified name and ID

`public String getName()`
Return name of this course

`public void setName(String name)`
Set name of this course

`public String getID()`
Return course ID of this course

6.6 TitoAnalyzer

public class TitoAnalyzer implements AnalyzerInterface

Class for analyzing tasks, required by eAssari.

Designed and implemented in 2nd development iteration.

6.7 TitoDisplayer

public class TitoDisplayer implements DisplayerInterface

Class for displaying tasks, required by eAssari.

Designed and implemented in 2nd development iteration.

6.8 TitoState

public class TitoState

Capsulates the end-state of single run of TitoKone.

This class will provide methods for querying the TitoKone end state.

Designed and implemented in 2nd development iteration.

7 JSP sivujen sekvenssikaaviot

Tämä luku esittelee toteutettavien JSP-sivujen toimintalogiikan. Sekvenssikaaviot esittelevät tärkeimmät eri toiminnoissa käytetyt luokat ja metodit, menemättä kuitenkaan aivan samalle tasolle kuin lopullinen ohjelmakoodi. Myöskään kaikkia toimintoja ei kuvata, esimerkiksi erilaisten listausten näyttäminen katsotaan niin suoraviivaiseksi, että ne onnistuvat ilman erillistä suunnitelmaa.

Sekvenssikaavioit esittävät miten TitoTrainer/eAssari käsittelee tapahtuman, jossa käyttäjä on täyttänyt lomakkeen ja lähettänyt sen palvelimen käsiteltäväksi. Vain onnistuneet tapahtumat on kuvattu, esimerkiksi väärän salasanan syöttäminen kirjautumislomakkeeseen katkaisee kuvatun sekvenssin ja palauttaa käyttäjälle virheilmoituksen.

7.1 Register.jsp

7.2 Login.jsp

7.3 TeacherTaskList.jsp

7.4 TaskList.jsp

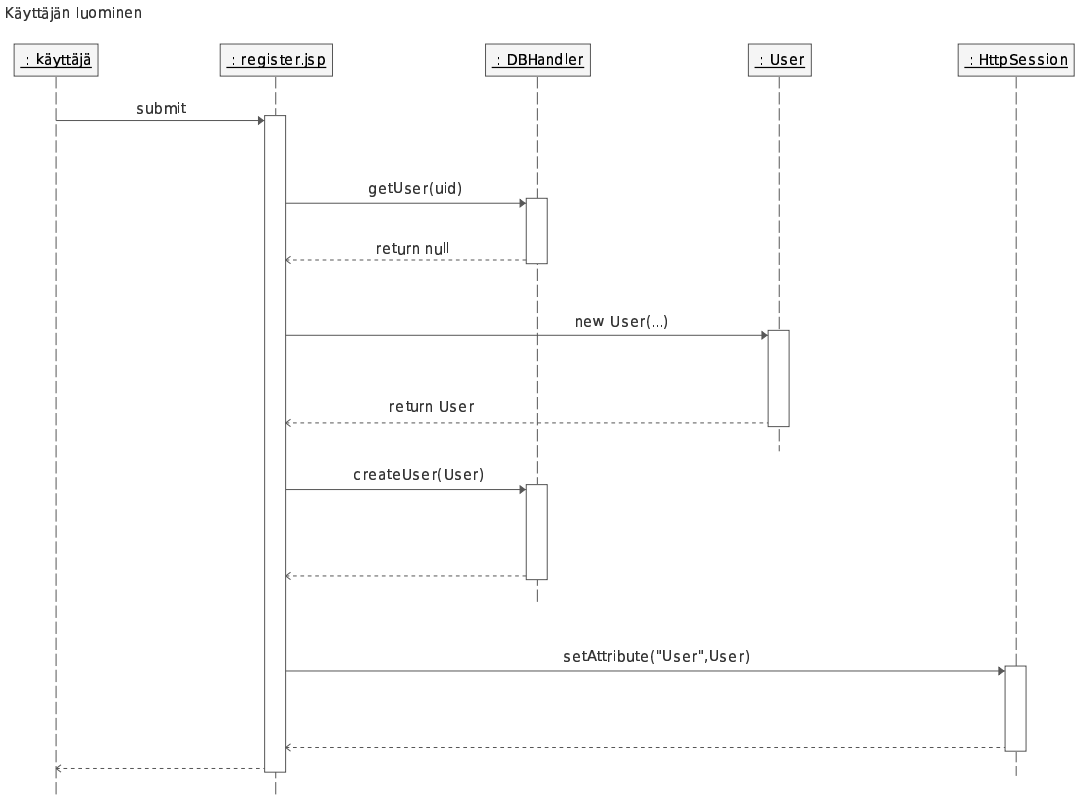
Ei kaaviota. Sisältää vain opiskelijan tehtävälistauksen.

7.5 Report.jsp

Toteutetaan ja suunnitellaan toisessa iteraatiossa

7.6 UserInfo.jsp

Toteutetaan ja suunnitellaan toisessa iteraatiossa



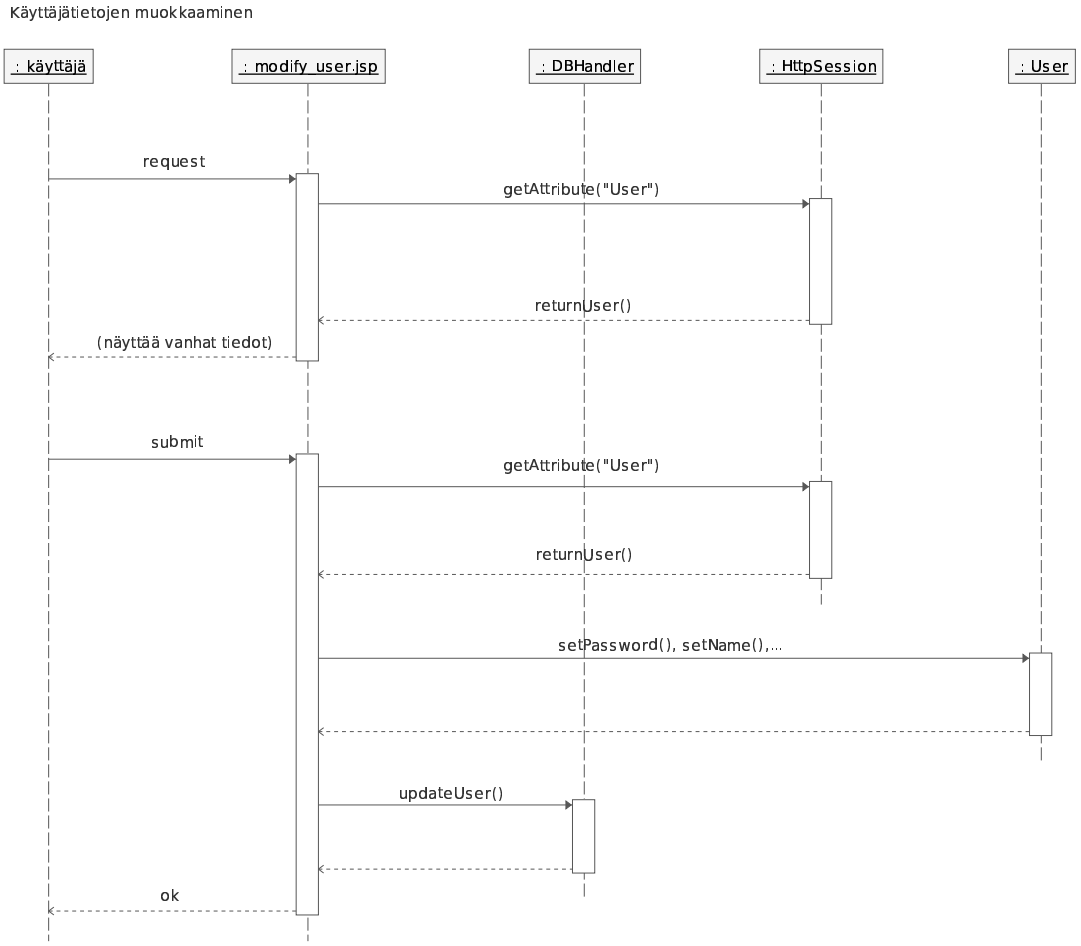
Kuva 2: Käyttäjän rekisteröityminen

7.7 Composer.jsp

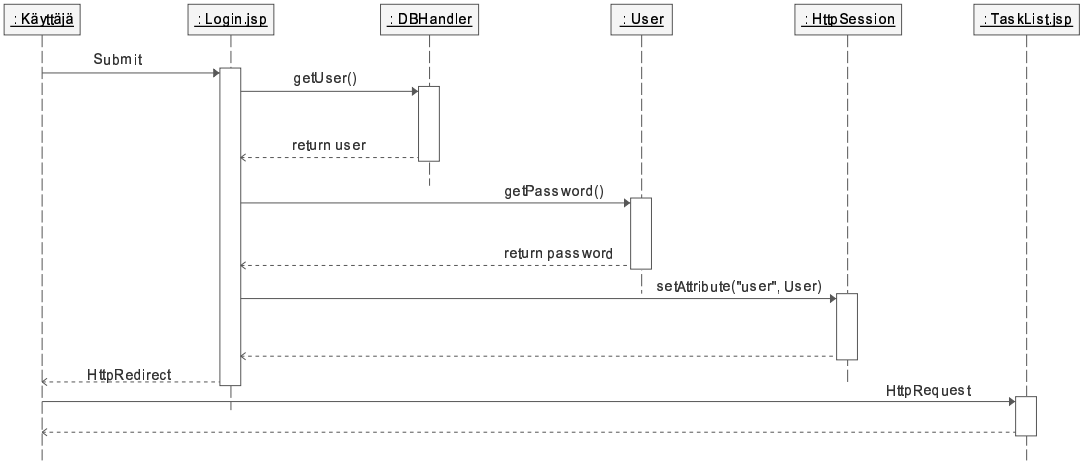
7.8 TeacherTaskList.jsp

7.9 Answer-servlet

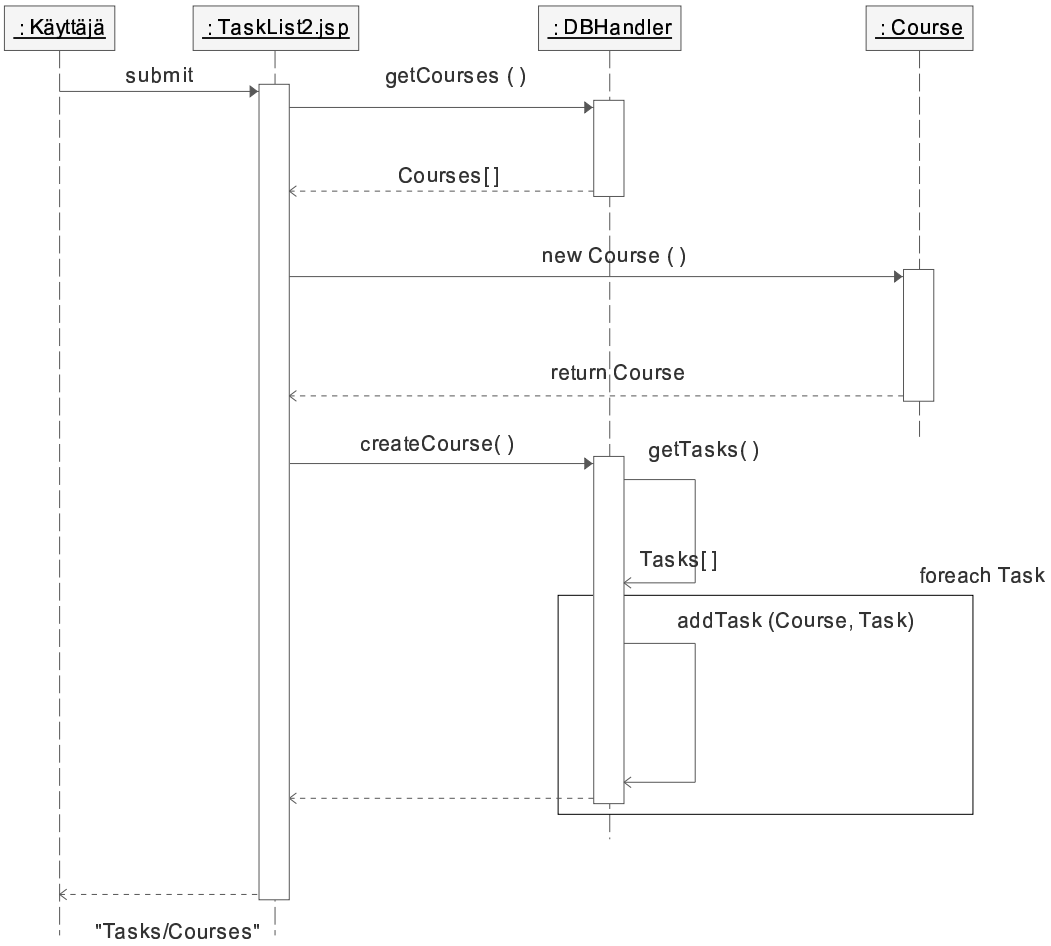
Toteutetaan ja suunnitellaan toisessa iteraatiossa



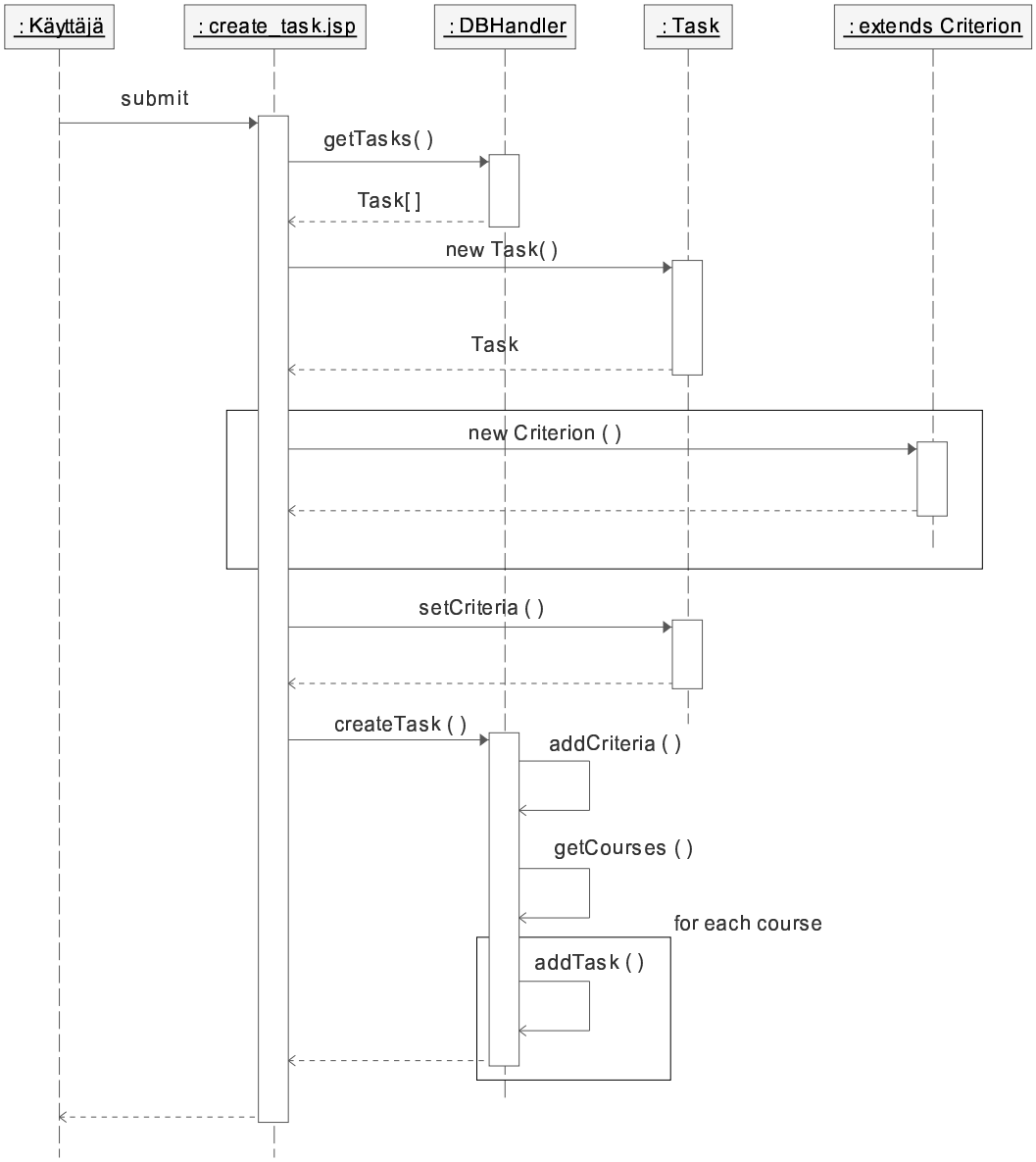
Kuva 3: Käyttäjätietojen muokkaaminen



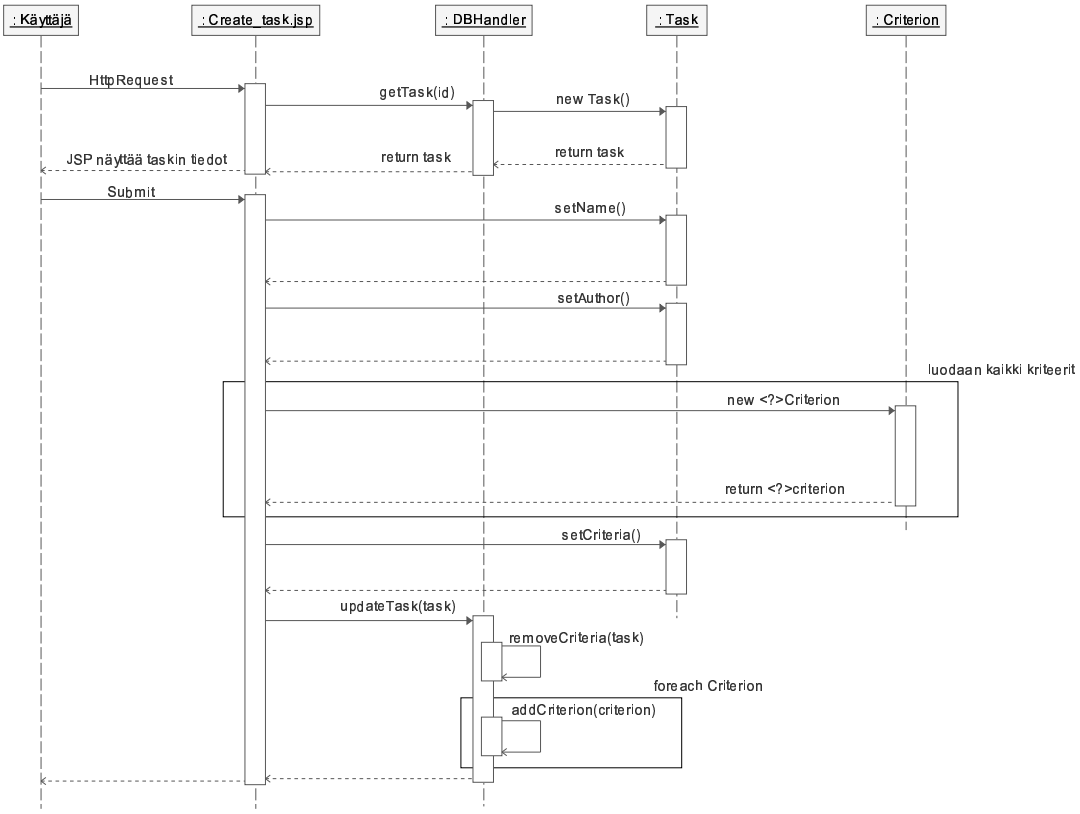
Kuva 4: Kirjautuminen



Kuva 5: Kurssin luominen



Kuva 6: Tehtävän luominen



Kuva 7: Tehtävän muokkaaminen